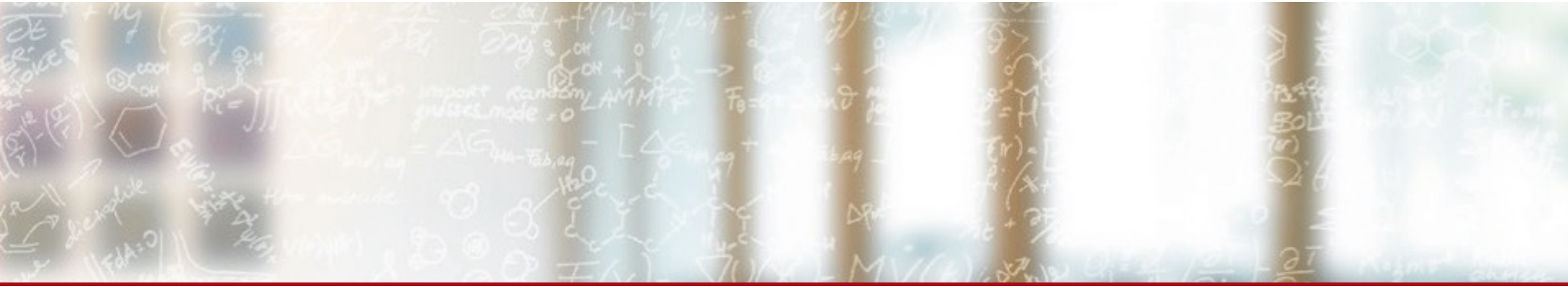
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Linear Algebra with C++ tasks: DLA-Future

Raffaele Solcà, CSCS

October 05, 2022

Motivation:

- In Density Functional Theory (DFT) applications, the Kohn-Sham equation

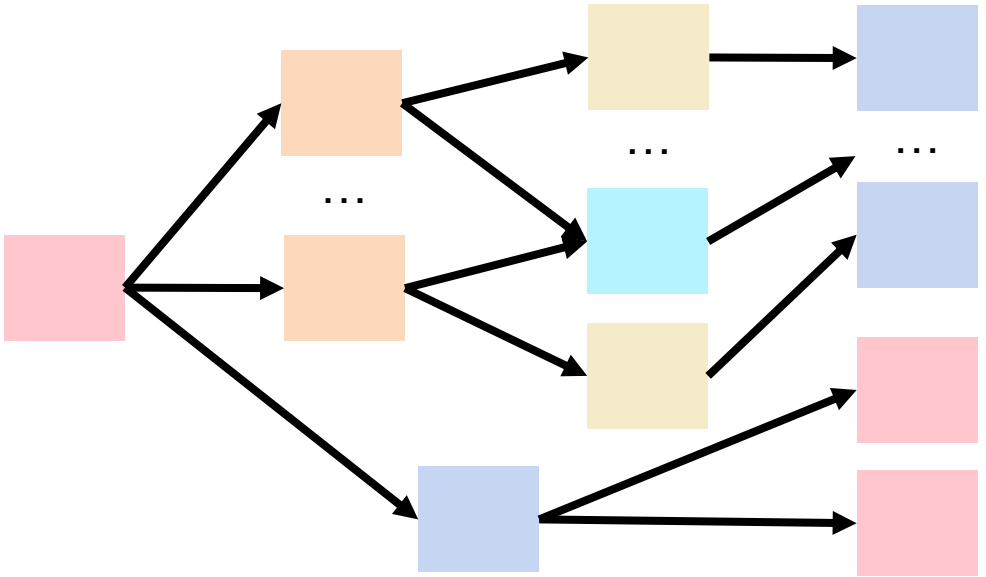
$$\left[\frac{\hbar}{2m}\nabla^2 + V_{eff}(\mathbf{r})\right]\psi_i(\mathbf{r}) = \epsilon_i\psi_i(\mathbf{r})$$

  has to be solved many times to find the ground state density functional.
- Introducing a basis set for the orbitals $\psi_i(\mathbf{r})$ the equation becomes the generalized eigenvalue problem $H\mathbf{x} = \epsilon\mathbf{O}\mathbf{x}$.
- The solution of the generalized Hermitian eigenvalue problem is the dominant part of the computation in material science.

Linear algebra parallelism

- ScaLAPACK and fork-join parallelism:
  - Work is sequential and low-level algorithms are multithread (BLAS)

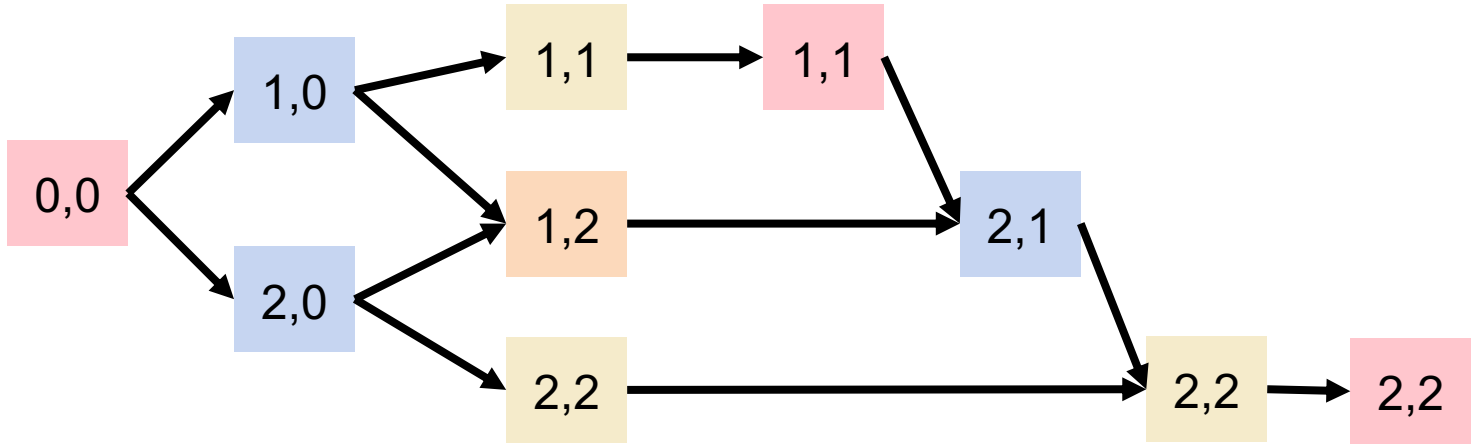
- Task based parallelism:



Tasks in linear algebra

- Sum $C_{ij} = A_{ij} + B_{ij}$:
  - For each element the operation is independent.
  - Can be easily expressed in tile form.

- (in-place) Cholesky for a Matrix:
  - 3x3 tiles:

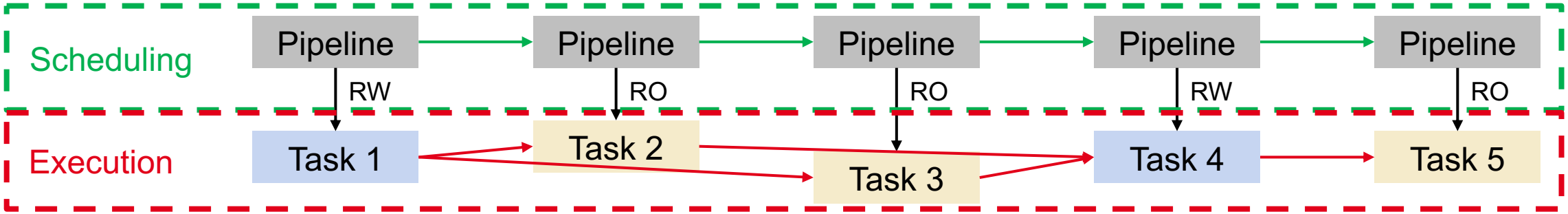


How DLA-Future has been designed

- Performance portable generalized eigenvalue solver for Hermitian matrices
- Use pika as tasking library to introduce fine-grained task dependencies:
  - allows starting the next algorithm even if the prev. has not finished,
  - allows to execute two independent algorithms at the same time,
  - Matrix Tiles are used to keep track of the dependencies of the various tasks.
- Use MPI as communication library:
  - allows use of the library in existing applications (C++, C, Fortran)
    (C and Fortran with ScaLAPACK-like interface).
- Use 2D block-cyclic distribution
  - compatible with ScaLAPACK

The Tile pipeline

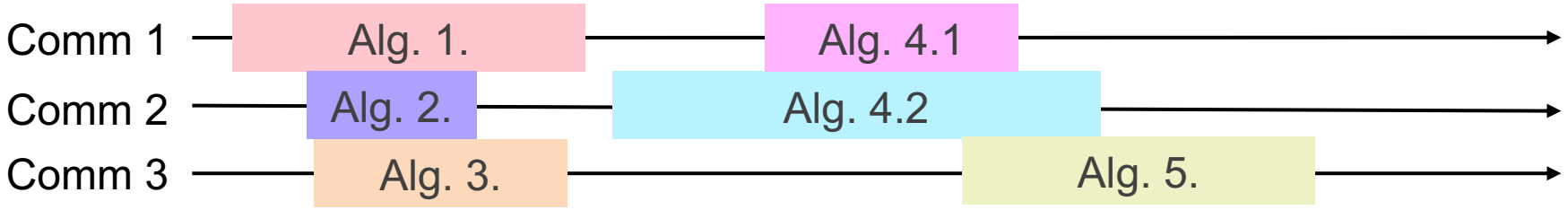
- Problem: management of the dependencies with senders is error prone.
  - Tiles might be read-only and read-write
  - Returning the tiles is error prone:
    - For read-only tiles all senders of the same tile has to be combined,
    - Returned tiles have to be stored correctly.
- Solution: Tile pipeline
  - Provides read-only and read-write senders.
    - When read-only, the elements of the Tile are constant.
  - The dependencies are triggered when the destructor of the Tile is invoked.



  - Note: Each tile gets an indipendent Pipeline

Communication

- Communication has to be correctly matched:
  - Point to point communication can use MPI tags
    - But need unique tags
  - Collectives needs ordering or multiple communicators
- Currently: duplicate communicator at each algorithm invocation
- Final solution: Have a queue of duplicated communicators which are assigned to algorithms
  - Async collectives scheduling is ordered on each communicator
  - Tags have to be unique within an algorithm



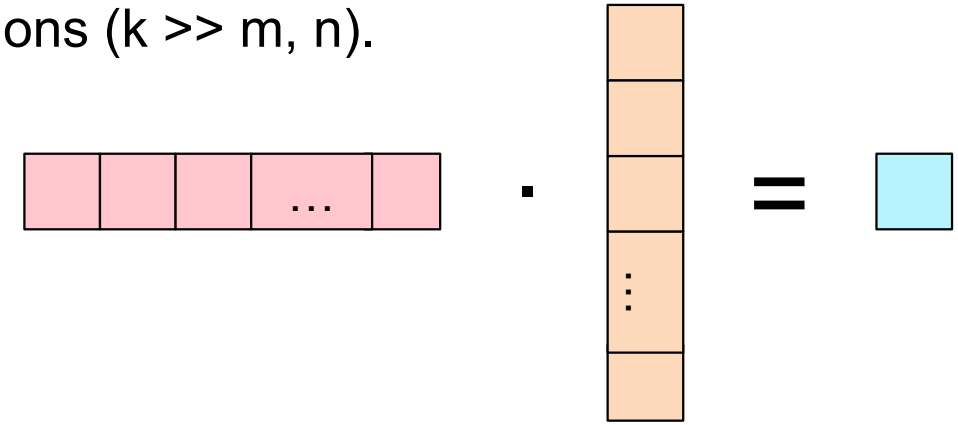
Memory usage

- Problem: Need some space for:
  - For temporary tiles needed in the computation,
  - To receive data from other MPI ranks.
- Simple solution to allocate the space when is needed is not ideal:
  - Memory usage might quickly explode (receive operations have no dependencies)
  - Allocation and deallocation generates GPU synchronizations.
- Solution: each algorithm allocates some space which is reused
  - Decrease memory usage
  - Decrease the number of pending asynchronous MPI receive.

  - Dimension of the space has to be carefully chosen as reusing it introduces non-real dependencies

Reductions

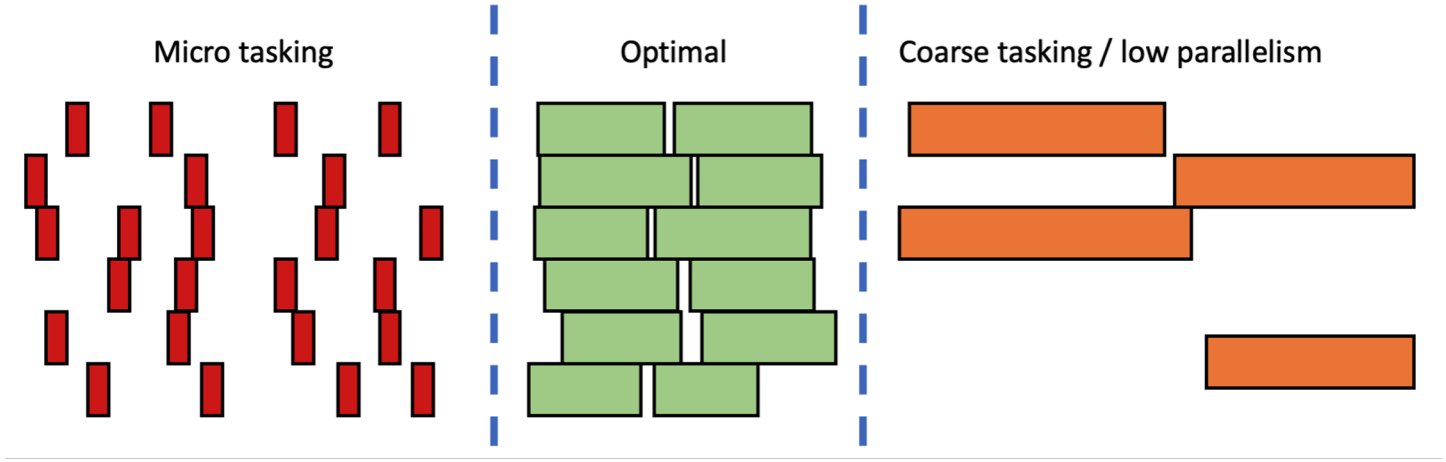
- Problem: multiple tasks which update the same result (value or tile) are serialized.
- Solution: introduce multiple workspace which are combined in the end.
- Examples:
  - Dot product of vectors, $\sum_i \mathbf{a}_i \mathbf{b}_i$
  - Frobenius norm of a matrix, $\sqrt{\sum_{i,j} |a_{ij}|^2}$
  - Tall and skinny matrix-matrix multiplications (k >> m, n).



Task sizes

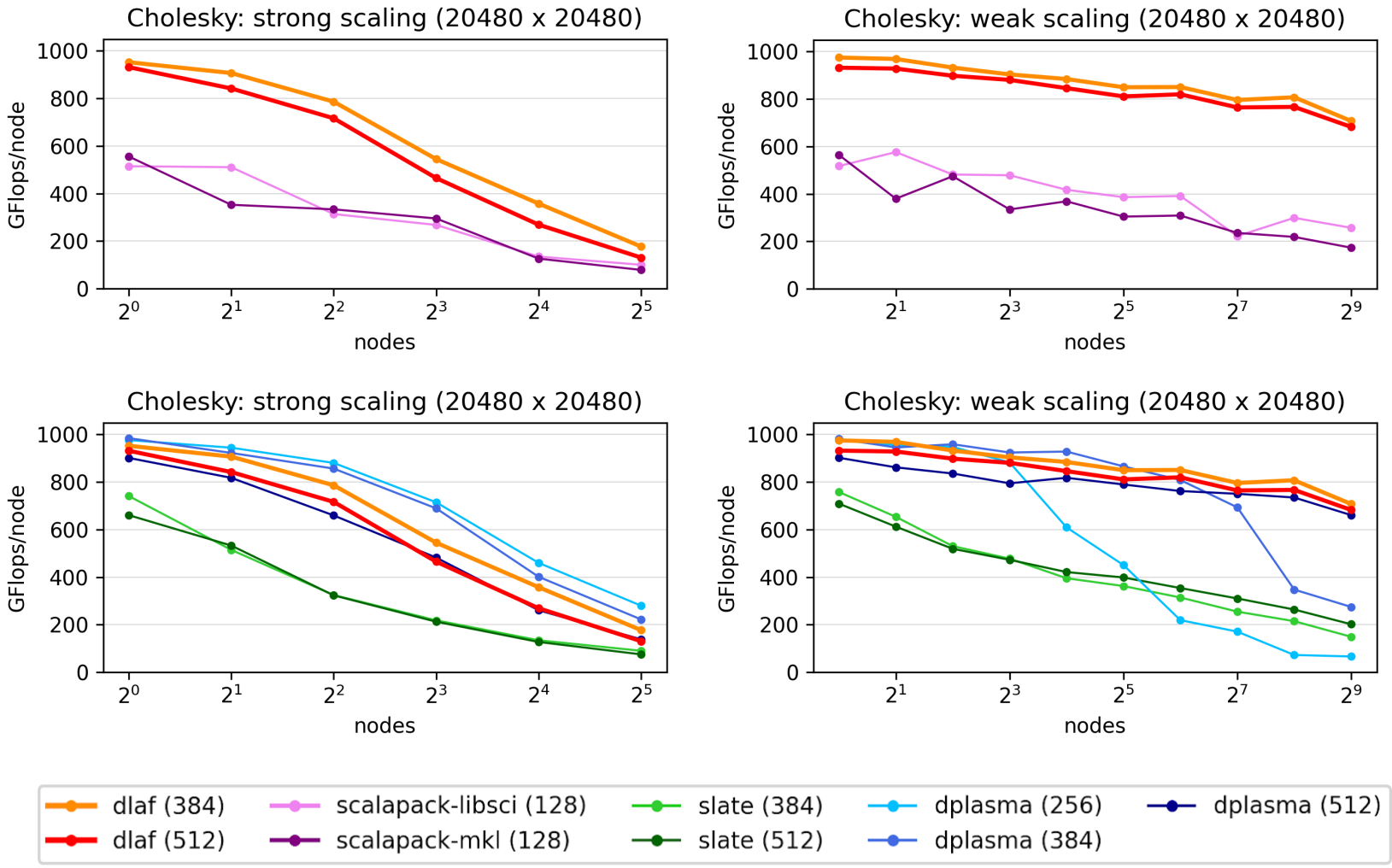
- Tasking runtimes introduce overheads in the execution.
  - context switching,
  - queue handling,
  - stack creation,

  contributes to a 1-10 µs overhead in pika.



- Need a minimum task size of 1 ms to achieve good efficiency.
- For some algorithms task sizes are comparable and adjusted with the tile size.
- For other algorithms small tasks must be combined into larger tasks affecting parallelism:
  - Less tasks are available
  - Extra dependencies are introduced

Cholesky: multicore results

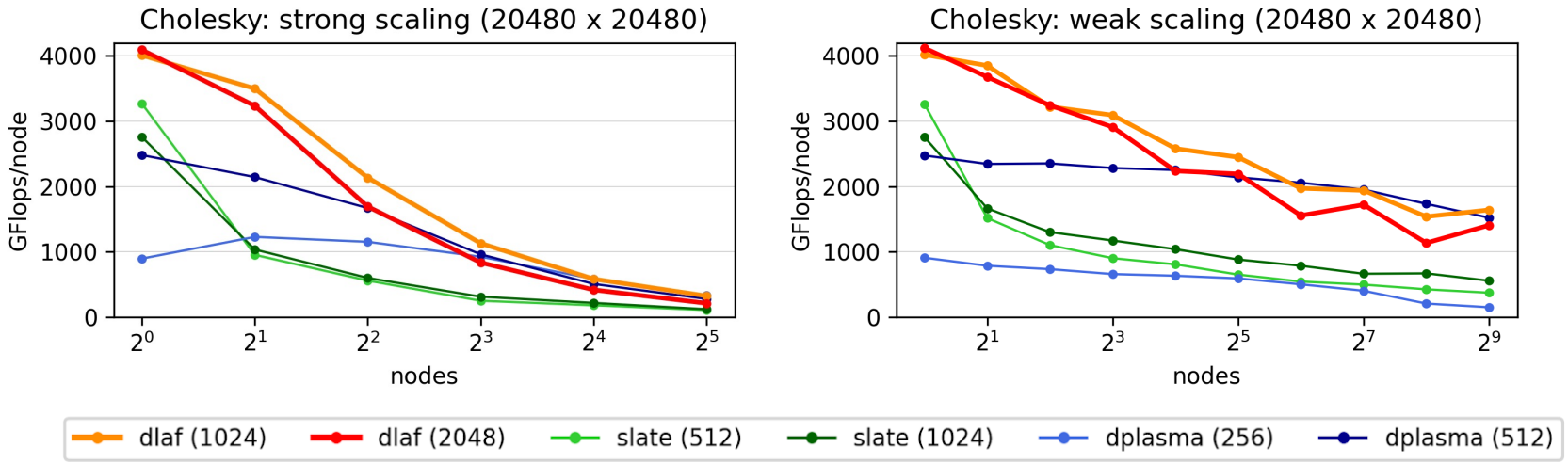


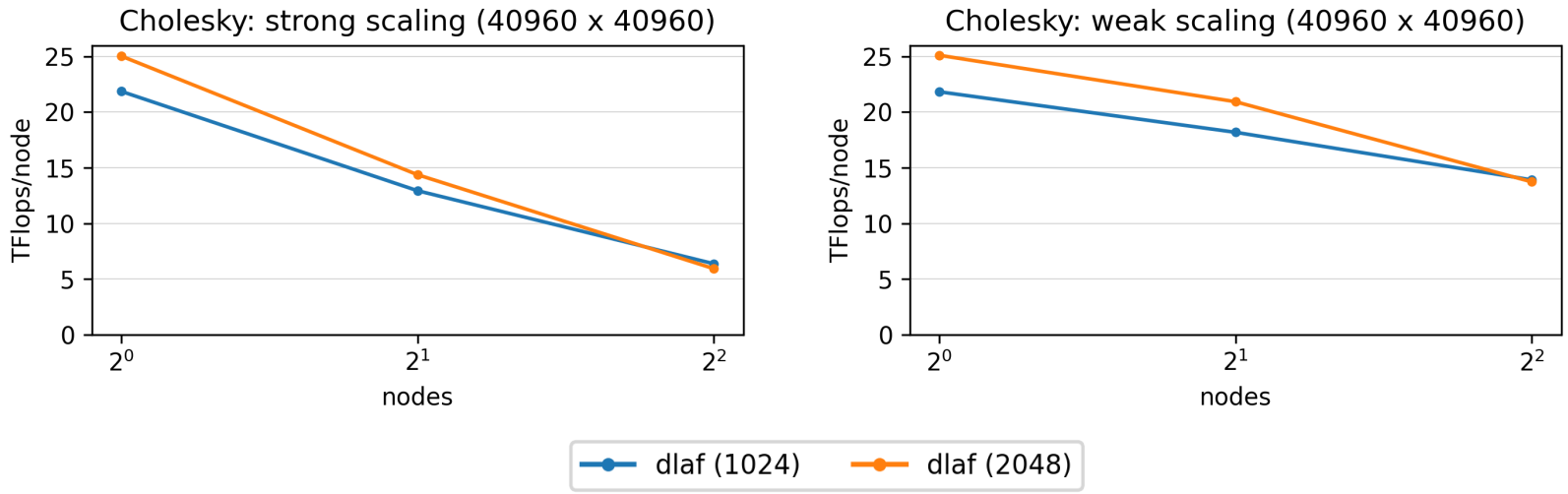
Results on Daint MC
(36 core Intel Broadwell)

Left: strong scaling for a matrix of size 20k.

Right: weak scaling
(20k matrix per node)

Cholesky: GPU results



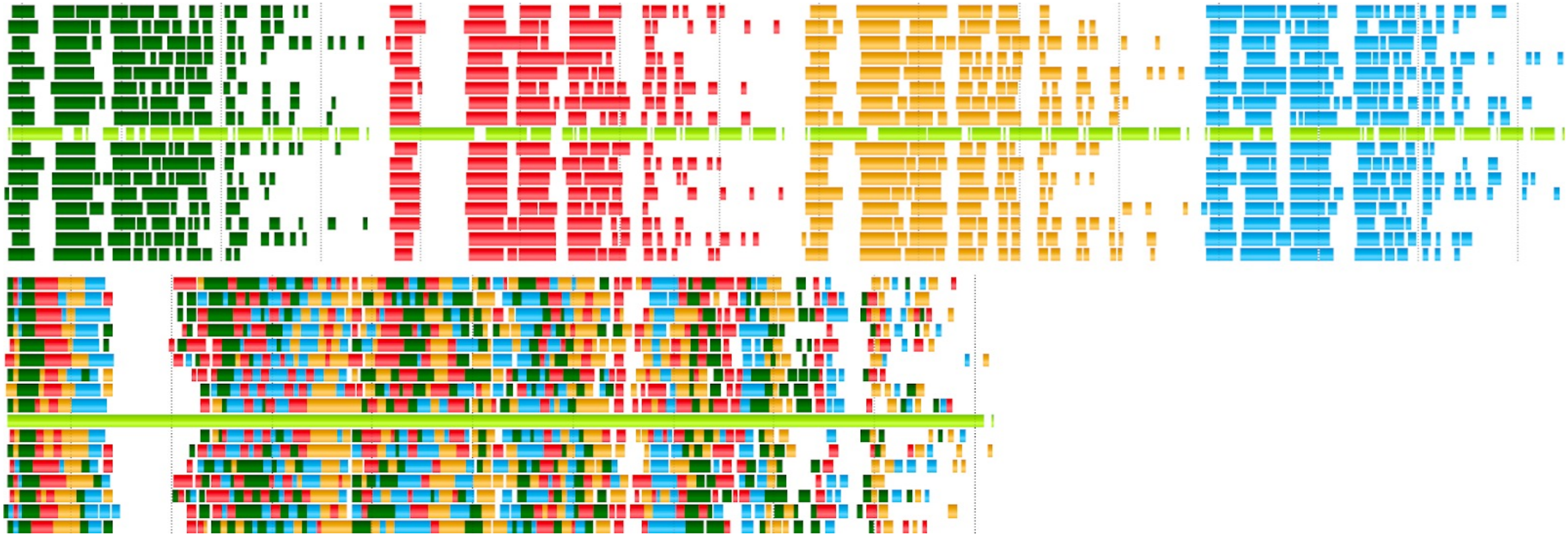


Upper:
Results on Daint GPU
(12 core Intel Haswell
+ Nvidia P100)
Left: strong scaling for a
matrix of size 20k.
Right: weak scaling
(20k matrix per node)

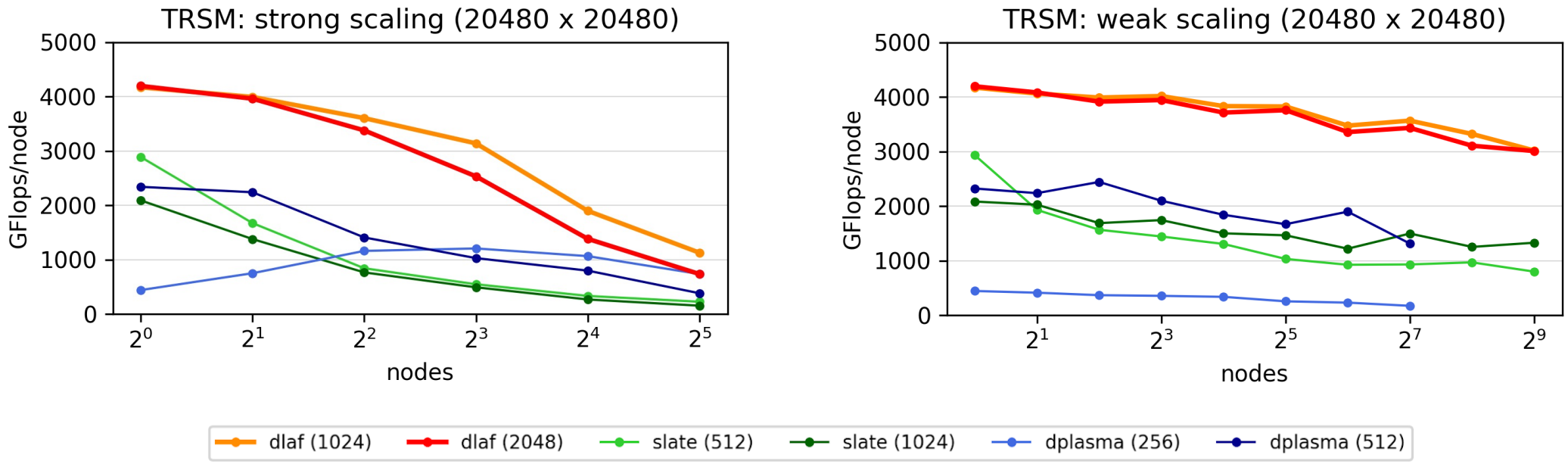
Lower:
Results on Alps
(64 core AMD Milan
+ 4x Nvidia A100)
Left: strong scaling for a
matrix of size 40k.
Right: weak scaling
(40k matrix per node)

Cholesky overlap

- Trace of 1 rank (over a total of 4) for 4 independent Cholesky decompositions. The tasks of each factorization have a different color. (MPI communications light-green). Above synchronized, below overlapped.

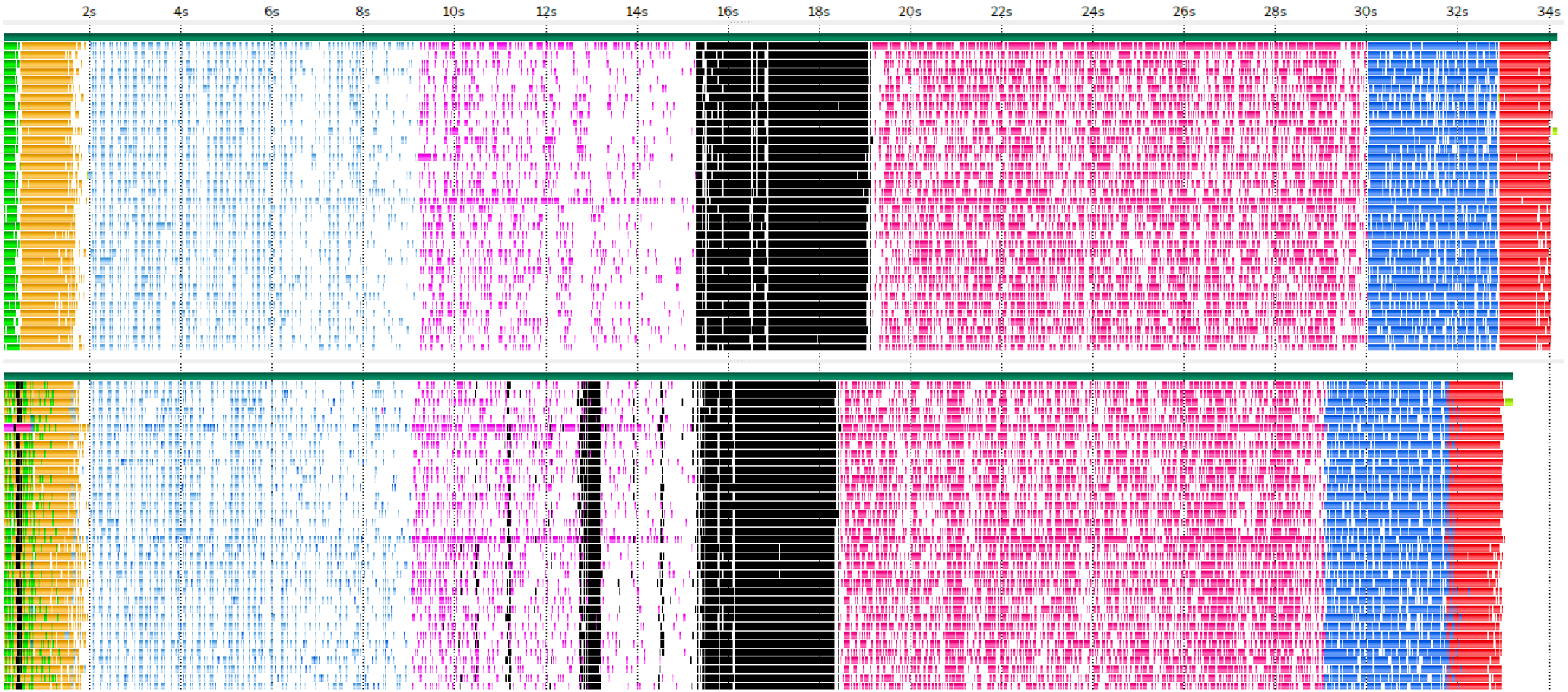


Triangular solver



Results on Daint GPU (12 core Intel Haswell + Nvidia P100)
Left: strong scaling for a matrix of size 20k.
Right: weak scaling (20k matrix per node)

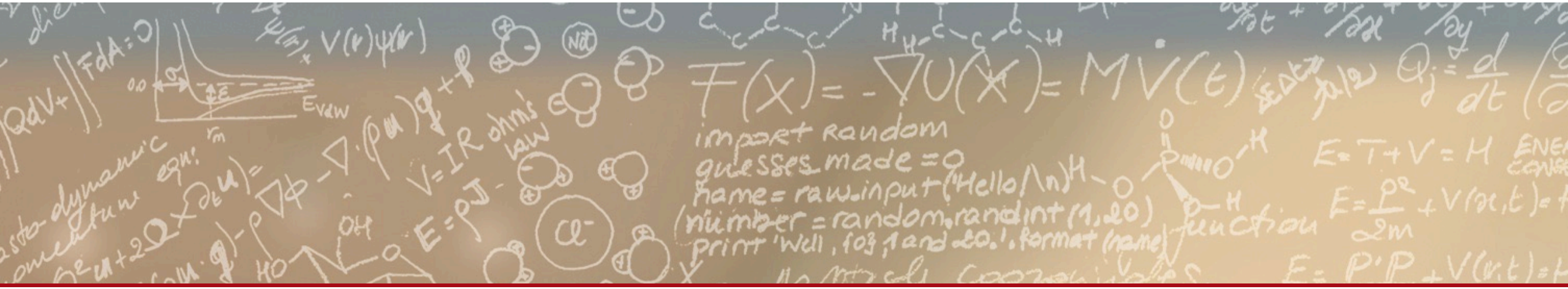
Eigensolver on single node Daint MC



Generalized eigensolver 10k matrix	DLA-Future	Intel MKL
Daint MC	32.9 s	27.4 s

DLAF status

- Current focus: algorithms needed by the eigensolver
- Completed: (single node / distributed, multicore / Cuda / ROCm)
  - Cholesky decomposition,
  - Triangular solver,
  - Triangular multiplication.
- Partially completed: (single node, multicore / Cuda)
  - Standard symmetric/Hermitian eigenvalue solver,
  - Generalized symmetric/Hermitian eigenvalue solver.



Thank you for your attention.

Questions?